Optimierung – Branch-and-Bound

Algorithmen und Datenstrukturen
VU 186.866, 5.5h, 8 ECTS, 2023S
Letzte Änderung: 22. Mai 2023

Vorlesungsfolien

ALGORITHMS AND COMPLEXITY GROUP
TU WIEN Informatics

Kombinatorische Optimierung

Kombinatorische Optimierung: Es geht bei der kombinatorischen Optimierung darum, aus einer (großen) Menge von diskreten Elementen (Gegenstände, Orte usw.) eine Teilmenge zu konstruieren,

- die gewissen Nebenbedingungen entspricht
- und bezüglich einer Kostenfunktion optimal ist (kleinstes Gewicht, kürzeste Strecken, . . .).

Kombinatorische Optimierungsaufgaben

Kombinatorische Optimierungsaufgaben: Wir haben bereits mehrere kombinatorische Optimierungsaufgaben in der Vorlesung betrachtet, z.B.

- Minimaler Spannbaum eines Graphen
- Kürzeste Wege in einem Graphen
- Minimales Vertex oder Set Cover
- Maximales Independent Set
- Minimale Knotenfärbung

Algorithmen-Paradigmen: Wir haben Methoden für das Lösen solcher Aufgaben kennengelernt, z.B. Greedy- Konstruktionsverfahren.

Hinweis: Greedy-Konstruktionsverfahren funktionieren aber nicht bei allen Problemen!

Optimierung: Schwere Probleme

Schwere Probleme: In dieser LVA wurden schon einige Probleme besprochen, für die es unwahrscheinlich ist, dass Lösungsverfahren existieren, die alle möglichen Instanzen eines bestimmten Problems in polynomieller Zeit lösen.

Anwendung: In dieser und den nachfolgenden Einheiten werden wir uns mit Verfahren beschäftigen, die bei solchen Problemen grundsätzlich angewendet werden können.

Verfahren für Optimierungsprobleme:

- Branch-and-Bound
- Dynamische Programmierung
- Approximation(salgorithmen)
- Heuristische Verfahren

Optimierung: Roadmap

Branch-and-Bound: Beschränke eine auf Divide-and-Conquer basierende systematische Durchmusterung aller Lösungen mit Hilfe von Methoden, die untere und obere Schranken liefern, und ermittle eine optimale Lösung.

Dynamische Programmierung

Approximation(salgorithmen)

Heuristische Verfahren

Überblick

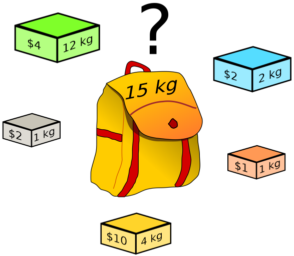
Rucksackproblem

Branch-and-Bound für das Rucksackproblem

Branch-and-Bound für Minimales Vertex Cover

Rucksackproblem

Gegeben: n Gegenstände mit positiven rationalen Gewichten $g_1, \ldots, g_n$ und Werten $w_1, \ldots, w_n$ und eine Kapazität G (auch positiv rational).



Gesucht: Teilmenge S der Gegenstände mit Gesamtgewicht $\leq G$ und maximalem Gesamtwert.

Rucksackproblem: Mathematische Formulierung

Entscheidungsvariablen: Wir führen 0/1-Entscheidungsvariablen für die Wahl der Gegenstände ein:

$$x_1, \ldots, x_n, \text{ wobei } x_i = \begin{cases} 0 & \text{falls Gegenstand } i \text{ nicht gewählt wird} \\ 1 & \text{falls Gegenstand } i \text{ gewählt wird} \end{cases}$$

Mathematische Formulierung: Für n Gegenstände:

Maximiere

$$\sum_{i=1}^{n} w_i x_i,$$

wobei

$$\sum_{i=1}^{n} g_i x_i \leq G, \quad x_i \in \{0, 1\}$$

Rucksackproblem: Enumeration (Backtracking)

Enumeration: Eine Enumeration aller zulässigen Lösungen für das Rucksackproblem entspricht der Aufzählung aller Teilmengen der n-elementigen Menge (bis auf diejenigen Teilmengen, die nicht in den Rucksack passen).

Lösungsvektor und Zielfunktion: Zu jedem aktuellen Lösungsvektor $\vec{x} = (x_1, \ldots, x_n)$ gehört ein Zielfunktionswert (Gesamtwert von $\vec{x}$) w_{curr} und ein Gesamtgewicht g_{curr}. Die bisher beste gefundene Lösung wird in dem globalen Vektor $\vec{x}_{\text{best}}$ und der zugehörige Lösungswert in der globalen Variablen $w_{\max}$ gespeichert.

Prinzip: Wir folgen wiederum dem Prinzip des Divide-and-Conquer.

Rucksackproblem: Enumerationsalgorithmus

Eingabe: Anzahl z der fixierten Variablen in $\vec{x}$; Gesamtwert w_{curr}; Gesamtgewicht g_{curr}; aktueller Lösungsvektor $\vec{x}$.

```
Enum(z, w_curr, g_curr, x⃗):
if g_curr ≤ G
    if w_curr > w_max
        w_max ← w_curr
        x⃗_best ← x⃗
    for i ← z + 1 bis n
        x_i ← 1
        Enum(i, w_curr + w_i, g_curr + g_i, x⃗)
        x_i ← 0
```

Hinweis:

- $w_{\max}$ und $\vec{x}_{\text{best}}$ sind globale Variablen.
- Initialisierung: $w_{\max} = 0$ und $\vec{x}_{\text{best}} = \vec{0}$

Rucksackproblem: Enumerationsalgorithmus

Ablauf: Der Algorithmus wird mit dem Aufruf Enum$(0, 0, 0, \vec{0})$ gestartet.

- In jedem rekursiven Aufruf wird die aktuelle Lösung $\vec{x}$ bewertet.
- Danach werden die Variablen x_1 bis x_z als fixiert betrachtet.
- Der dadurch beschriebene Teil des gesamten Suchraums wird weiter unterteilt: Wir betrachten alle möglichen Fälle, welche Variable x_i (mit $i = z + 1$ bis $i = n$) als nächstes auf 1 gesetzt werden kann.
- Die Variablen x_{z+1} bis x_{i-1} werden gleichzeitig auf 0 fixiert.
- Alle so erzeugten kleineren Unterprobleme werden durch rekursive Aufrufe gelöst.

Komplexität: Es gibt bis zu 2^n rekursive Aufrufe und der Aufwand pro Aufruf (exklusive Rekursion) ist konstant. Daher liegt die Laufzeit in $O(2^n)$.

Branch-and-Bound

Rucksackproblem: Verbesserung der Enumeration

Idee zur Verbesserung: Überprüfen von Zwischenlösungen mit $z < n$ fixierten Variablenwerten.

- Man überprüft, ob es noch möglich sein kann, aus dieser Lösung durch Hinzufügen weiterer Gegenstände eine zu erzeugen, die besser ist als die bisher beste gefundene.
- Wenn es offensichtlich ist, dass keine neue beste Lösung abgeleitet werden kann, dann sind weitere rekursive Aufrufe nicht sinnvoll.
- Das frühzeitige Abbrechen führt zu einer Beschneidung des rekursiven Aufrufbaums.
- Das kann eine erhebliche Beschleunigung bewirken.

Branch-and-Bound: Rucksackproblem

Ansatz:

- Berechne obere Schranke U', und führe den Aufruf nur durch, wenn der Wert $U' > w_{\max}$.
- Sortiere die Gegenstände nach nicht-steigenden Werten $\frac{w_i}{g_i}$.

```
Enum(z, w_curr, g_curr, x⃗):
if g_curr ≤ G
    if w_curr > w_max
        w_max ← w_curr
        x⃗_best ← x⃗
    for i ← z + 1 bis n
        U' ← w_curr + (G − g_curr) · w_i/g_i
        if U' > w_max
            x_i ← 1
            Enum(i, w_curr + w_i, g_curr + g_i, x⃗)
            x_i ← 0
```

Branch-and-Bound: Rucksackproblem

Obere Schranke: $U' \leftarrow w_{\text{curr}} + (G - g_{\text{curr}}) \cdot \frac{w_i}{g_i}$

Erklärung:

- w_{curr} enthält den Wert der bisherigen Zuteilung.
- $G - g_{\text{curr}}$ ist die verbleibende Kapazität im Rucksack.
- Die verbleibende Kapazität wird mit dem aktuell untersuchten Gegenstand i komplett (vielleicht auch mehrmals) aufgefüllt. Hierbei kann es auch zu teilweisen Zuteilungen kommen (z.B. Gegenstand i wird 1,7 mal eingepackt).
- Da die Gegenstände nach nicht-steigenden Werten $\frac{w_i}{g_i}$ sortiert sind, haben alle Gegenstände $i+1, i+2, ..., n$ einen relativen Wert kleiner als der von i.
- Damit wird die obere Schranke U' garantiert größer gleich dem Wert der optimalen Lösung sein (für dieses Teilproblem).

Prinzip von Branch-and-Bound: Maximierungsproblem

Branching: Wie bei der Enumeration üblich wird das Problem rekursiv in kleinere Teilprobleme partitioniert $\rightarrow$ *Divide-and-Conquer*-Prinzip.

Bounding: Für jedes Teilproblem wird

- eine lokale obere Schranke U' (*upper bound*) und
- eine lokale untere Schranke L' (*lower bound*)

berechnet.

Abbruch: Teilprobleme mit $U' \leq L$ (L entspricht einer globalen unteren Schranke) brauchen nicht weiter verfolgt werden!

Schranken:

- Der Wert jeder gültigen Lösung ist eine untere Schranke.
- Obere Schranken werden i. A. separat mit einer sogenannten *Dualheuristik* ermittelt.

Rucksackproblem: Verbesserte untere Schranke

Sortierung: Die Gegenstände werden nicht-steigend nach ihren Werten $\frac{w_i}{g_i}$ sortiert.

Untere Schranke: Man durchläuft alle Gegenstände, deren Variablen noch nicht festgelegt sind, in der sortierten Reihenfolge und packt den jeweils aktuellen Gegenstand ein, falls noch Platz im Rucksack ist. (Greedy-Algorithmus)

Rucksackproblem: Verbesserte obere Schranke

Einfache obere Schranke wurde weiter vorne beschrieben.

Mögliche Verbesserung:

- Alle Gegenstände, deren Variablen noch nicht festgelegt sind, werden in der sortierten Reihenfolge durchlaufen.
- Man packt alle Gegenstände ein, bis man zu dem ersten Gegenstand kommt, der nicht mehr in den Rucksack passt.
- Sei r die noch freie Kapazität des Rucksacks. Dann zählt man $r \cdot w_i/g_i$ noch zu dem Wert der Gegenstände im Rucksack dazu.
- Der letzte Gegenstand wird daher nur teilweise eingepackt.
- Alle verbleibenden Gegenstände werden ignoriert.

Lösung:

- Diese Vorgehensweise liefert in der Regel eine Schranke, der keine gültige Lösung des Rucksackproblems entspricht.
- Falls diese Vorgehensweise zu einer gültigen Lösung führt, dann ist die Lösung (für das betrachtete Teilproblem) optimal.

Maximierungsproblem: Vorgehen

Allgemeines Vorgehen:

- Das Problem wird z.B. durch das Fixieren von Variablen oder Hinzufügen von Randbedingungen in Unterprobleme zerteilt, d.h. der Lösungsraum wird partitioniert.
- Ist für eine (oder mehrere) dieser Teilmengen die für sie berechnete obere Schranke U' nicht größer als die beste überhaupt bisher gefundene untere Schranke L ($=$ Wert der bisher besten Lösung), braucht man die Lösungen in dieser Teilmenge nicht mehr beachten.
- Ist die obere Schranke größer als die beste gegenwärtige untere Schranke, muss man die Teilmengen weiter zerkleinern.
- Man fährt solange mit der Zerteilung fort, bis für alle Lösungsteilmengen die obere Schranke nicht mehr größer ist als die (global) beste untere Schranke.

Maximierungsproblem: Allgemeiner Algorithmus

Eingabe: Instanz I

```
Branch-and-Bound-Max(I):
L ← −∞ oder Wert einer initialen heuristischen Lösung
Π ← {I}
while ∃I' ∈ Π
    Entferne I' aus Π
    Berechne für I' lokale obere Schranke U' mit Dualheuristik
    if U' > L
        Berechne für I' gültige heuristische Lösung → untere Schranke L'
        if L' > L
            L ← L'
        if U' > L
            Partitioniere I' in Teilinstanzen I_1, ..., I_k
            Π = Π ∪ {I_1, ..., I_k}
return beste gefundene Lösung mit Wert L
```

- *Bounding – Fall $U' \leq L$ nicht weiter interessant.*
- *Branching.*

Maximierungsproblem: Allgemeines Verfahren

Allgemeines Verfahren:

- Branch-and-Bound ist ein allgemeines Prinzip (Metaverfahren).
- Es kann auf verschiedenste diskrete Optimierungsprobleme angewendet werden.
- Entscheidend für die Effizienz ist
  - vor allem die Wahl der Heuristiken für U' und L',
  - wie das Branching erfolgt und
  - welche Teilinstanz ausgewählt wird.

Rucksackproblem: Beispiel

Gegeben: 4 Gegenstände, Rucksackkapazität $= 100$

Gegenstand	1	2	3	4
Gewicht g_i	32	16	21	50
Wert w_i	80	20	63	100
Verhältnis w_i/g_i	2.5	1.25	3	2

Sortierung:

- Für jeden Gegenstand i das Verhältnis w_i/g_i berechnen.
- Sortierte Reihenfolge der Gegenstände: 3 (3), 1 (2.5), 4 (2), 2 (1.25)

Rucksackproblem: Beispiel

Branch-and-Bound-Baum:

4 Gegenstände, Rucksackkapazität = 100

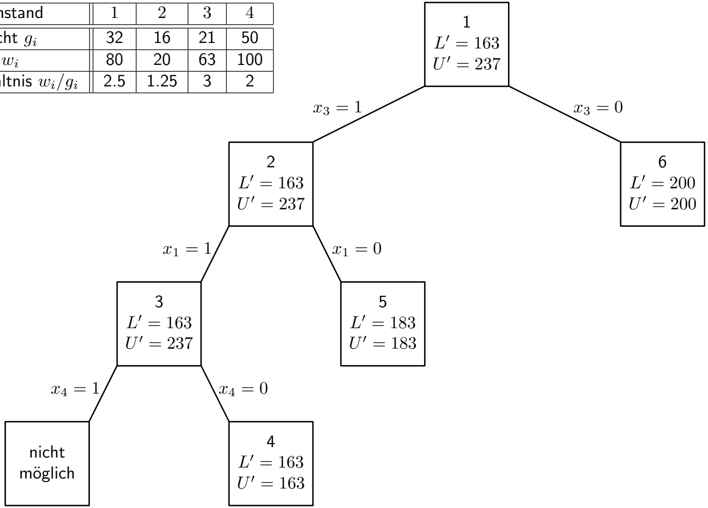
Gegenstand	1	2	3	4
Gewicht g_i	32	16	21	50
Wert w_i	80	20	63	100
Verhältnis w_i/g_i	2.5	1.25	3	2

Rucksackproblem: Beispiel

Branch-and-Bound-Baum:

4 Gegenstände, Rucksackkapazität = 100

Gegenstand	1	2	3	4
Gewicht g_i	32	16	21	50
Wert w_i	80	20	63	100
Verhältnis w_i/g_i	2.5	1.25	3	2



Rucksackproblem: Beispiel

Erklärung:

- In 1 (Start) sind noch keine Variablen fixiert.
- In 2 geht man davon aus, dass Gegenstand 3 (x_3) fixiert ist und nur die anderen Gegenstände ausgewählt werden können.
- Eine Fixierung von Gegenstand 3, 1 und 4 führt zu einer unmöglichen Lösung (würde eine Kapazität von 103 benötigen)
- Bei 4 und 5 ist $L = U$ und daher brauchen in diesem Unterbaum keine weiteren Gegenstände hinzugefügt werden (Beschneiden des rekursiven Aufrufbaums).
- Bei 6 ist auch $L = U$ (der Unterbaum braucht nicht mehr untersucht werden) und L ist hier am größten. Daher werden Gegenstand 1, 2 und 4 eingepackt ($x_3 = 0$).

Branch-and-Bound: Auswahl des nächsten Teilproblems

Auswahl des nächsten Teilproblems:

- Welches Teilproblem aus der Liste der offenen Probleme jeweils als nächstes ausgewählt und bearbeitet wird, ist für die grundsätzliche Funktionsweise und Korrektheit von Branch-and-Bound egal.
- Die Auswahl hat jedoch mitunter starke Auswirkungen auf die praktische Laufzeit.

Beispiele für Strategien:

- Best-first
- Depth-first

Branch-and-Bound: Auswahl der Probleme

Best-first:

- Es wird jeweils ein Teilproblem mit der besten dualen Schranke (also der größten oberen Schranke) ausgewählt.
- Dadurch wird immer die kleinstmögliche Anzahl an Teilproblemen abgearbeitet.

Depth-first:

- Es wird jeweils ein zuletzt erzeugtes Teilproblem weiter bearbeitet (vergleiche Tiefensuche bei der Durchmusterung von Graphen).
- Man erhält meist am raschesten eine vollständige und gültige Näherungslösung.
- Häufig wird auch mit einer Depth-first Strategie begonnen und nach Erhalt einer gültigen Lösung mit Best-first fortgesetzt um die Vorteile zu kombinieren.

Branch-and-Bound für Minimales Vertex Cover

Minimierungsproblem: Allgemeiner Algorithmus

Eingabe: Instanz I

```
Branch-and-Bound-Min(I):
U ← ∞ oder Wert einer initialen heuristischen Lösung

Π ← {I}
while ∃I' ∈ Π
    Entferne I' aus Π
    Berechne für I' lokale untere Schranke L' mit Dualheuristik
    if L' < U
        Berechne für I' gültige heuristische Lösung → obere Schranke U'
        if U' entspricht einer gültigen Lösung für I'
            if U' < U
                U ← U'
        if L' < U
            Partitioniere I' in Teilinstanzen I_1, ..., I_k
            Π ← Π ∪ {I_1, ..., I_k}
return beste gefundene Lösung mit Wert U
```

■ *Bounding - Fall $L' \geq U$ nicht weiter interessant.* ■ *Branching.*

Branch-and-Bound: Minimales Vertex Cover

Eingabe: Graph $G = (V, E)$ und Knotenmenge $C = \emptyset$

```
MinVertexCover-BranchAndBound(G, C):
U ← |V| − 1
Π ← {(G, C)}
while ∃I′ ∈ Π
    Entferne I′ aus Π
    Berechne für I′ = (G′, C′) lokale untere Schranke L′ mit Matchingheuristik
    if L′ < U
        Berechne für I′ gültige heuristische Lösung mit Greedyheuristik
            → obere Schranke U′
        if U′ < U
            U ← U′
        if L′ < U
            u_max ← Knoten mit maximalem Grad in G′
            Erzeuge Teilinstanzen I_1 = (G′ − {u_max}, C ∪ {u_max}) und
                I_2 = (G′ − {u_max} − N(u_max), C ∪ N(u_max))
            Π ← Π ∪ {I_1, I_2}
return beste gefundene Lösung mit Wert U
```

■ *alle Nachbarknoten von* $u_{\max}$

Branch-and-Bound: Minimales Vertex Cover

Untere Schranke: Wird mit Hilfe eines *Matchings* bestimmt.

- Sei ein Graph $G = (V, E)$ gegeben.
- Eine Menge $M \subseteq E$ heißt Matching, wenn keine zwei Kanten aus M einen Knoten gemeinsam haben.

Nicht erweiterbares Matching:

- Nicht erweiterbares Matching (*maximales Matching*) bedeutet, dass es keine Kante $e \in E \setminus M$ gibt, sodass $\{e\} \cup M$ ein gültiges Matching ist.
- Das ist nicht notwendigerweise ein größtes Matching.
- Nicht erweiterbares Matching kann mit einem Greedy-Verfahren gefunden werden.

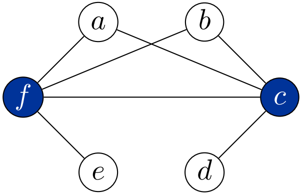
Branch-and-Bound: Minimales Vertex Cover

Berechnung der unteren Schranke L' für die Instanz (G', C'):

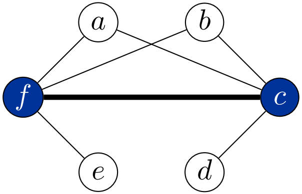
- Man wählt für L' die Größe eines nicht erweiterbaren Matchings.
  - Dabei wird zunächst eine beliebige Kante $e = (u, v)$ gewählt und dann die Knoten u und v und ihre inzidenten Kanten aus G' entfernt.
  - Man fährt mit dieser Prozedur fort, bis keine Kante mehr vorhanden ist.
  - Die Anzahl der gewählten Kanten entspricht der Größe des Matchings.
- Kanten in einem Matching haben keine Knoten gemeinsam.
- Ein Vertex Cover muss zumindest einen Knoten für jede Kante in einem Matching wählen.
- Daher ist die Größe eines Matchings von G' plus die Größe von C' eine untere Schranke für die Größe eines Vertex Covers des Eingabegraphen G.

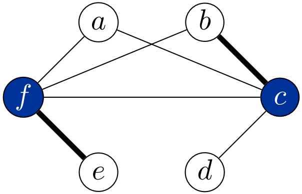
Branch-and-Bound: Beispiel für untere Schranke

Vertex Cover: Vertex Cover mit $k = 2$



Greedy-Matching: 2 Beispiele für Greedy-Matching (fett eingezeichnet Kanten).



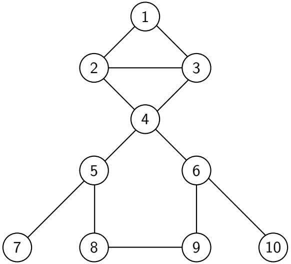


Branch-and-Bound: Minimales Vertex Cover

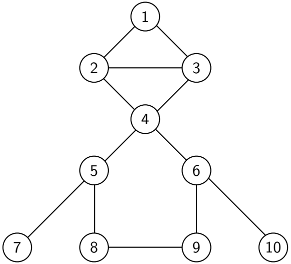
Obere Schranke U': Wird mit Hilfe eines Greedy-Algorithmus bestimmt.

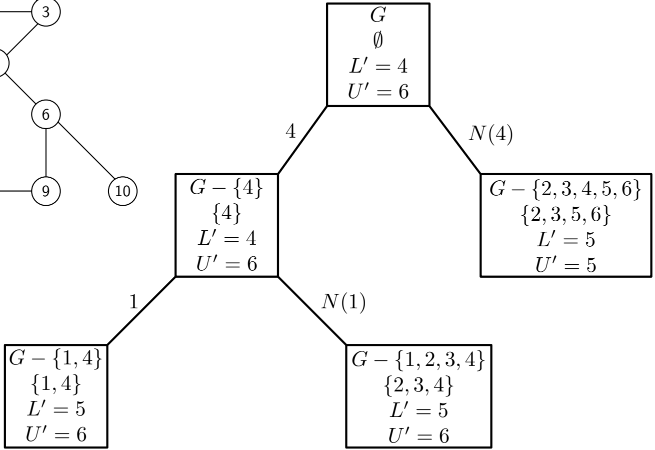
- Sei ein Graph $G' = (V', E')$ und eine Knotenmenge C' gegeben.
- Initialisiere eine Menge $S \leftarrow \emptyset$.
- Sortiere die Knoten nicht-steigend nach dem Knotengrad.
- Durchlaufe V' in dieser Reihenfolge solange der Graph noch Kanten enthält.
  - Füge den Knoten u mit höchstem Knotengrad zu S hinzu.
  - Entferne u und alle seine inzidenten Kanten aus G'.
  - Passe die Reihenfolge der verbleibenden Knoten an.
- Die Menge S ist ein Vertex Cover für G'.
- Daher ist $|C'| + |S|$ eine obere Schranke für die Größe eines minimalen Vertex Covers des Eingabegraphen G.

Minimales Vertex Cover: Beispiel



Minimales Vertex Cover: Beispiel





Branch-and-Bound: Zusammenfassung

- Branch-and-Bound ist eine allgemein für kombinatorische Optimierungsprobleme einsetzbare Technik.
- Sie funktioniert sowohl für Maximierungs- als auch für Minimierungsprobleme.
- Praktisch lassen sich oft hohe Beschleunigungen erreichen, die worst-case Laufzeit bleibt jedoch wie bei der Enumeration.

Vorgehen beim Entwurf von Branch-and-Bound Algorithmen:

- Wie lassen sich (Teil-)Instanzen des Problems ausdrücken?
- Was sind gute Heuristiken für untere und obere Schranken?
- Wie wird eine (Teil-)Instanz in weitere Teilinstanzen partitioniert (Branching)?
- Welche Teilinstanz wird im nächsten Schritt ausgewählt?